Common Compiler Infrastructure

Herman Venter
Microsoft Corporation

Among other things, it is yet another compiler compiler

X# Syntax and Semantics

Common Compiler Infrastructure

Not quite a semester project for undergrads

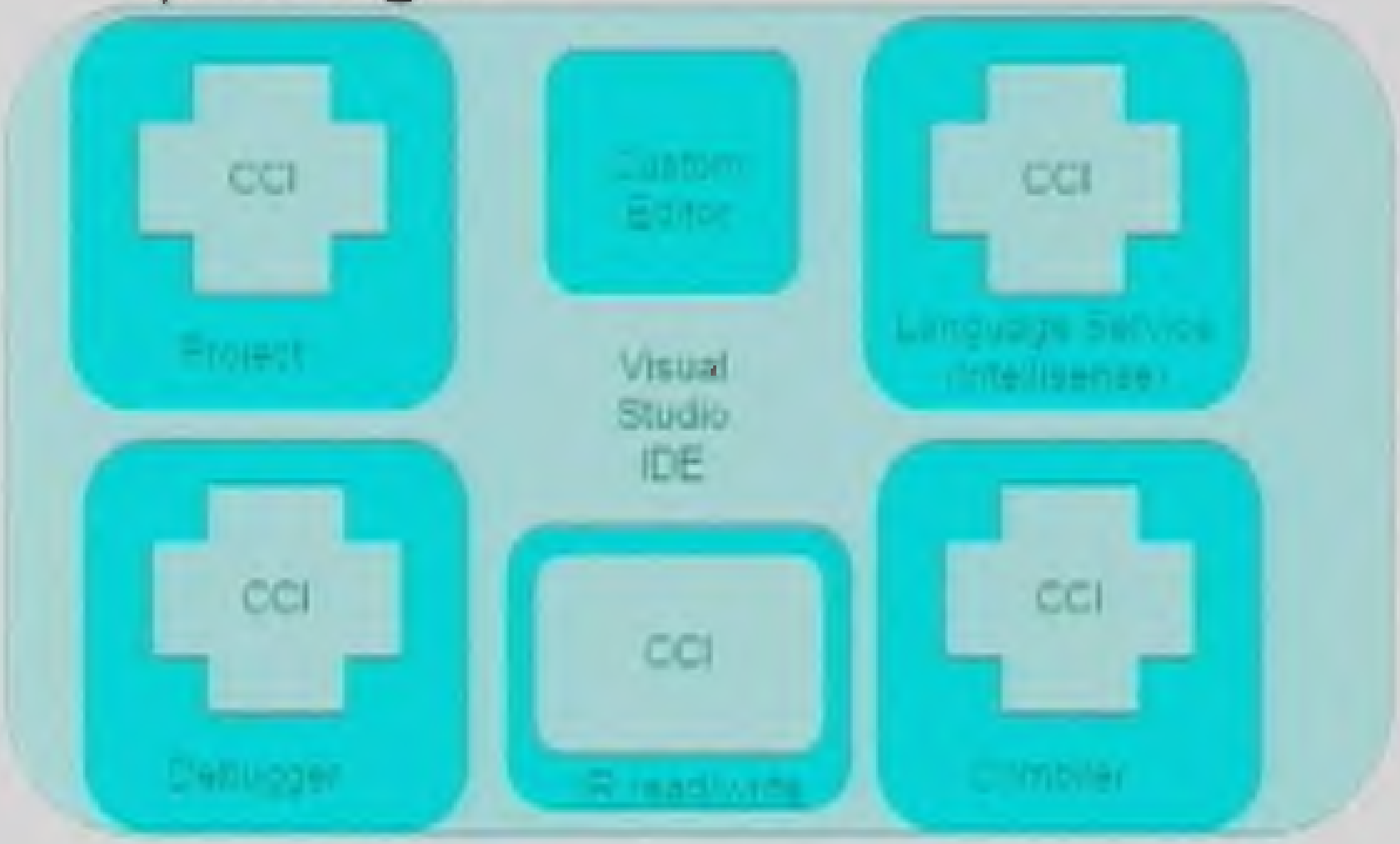
Among other things, it is yet another compiler compiler

X# Syntax and Semantics

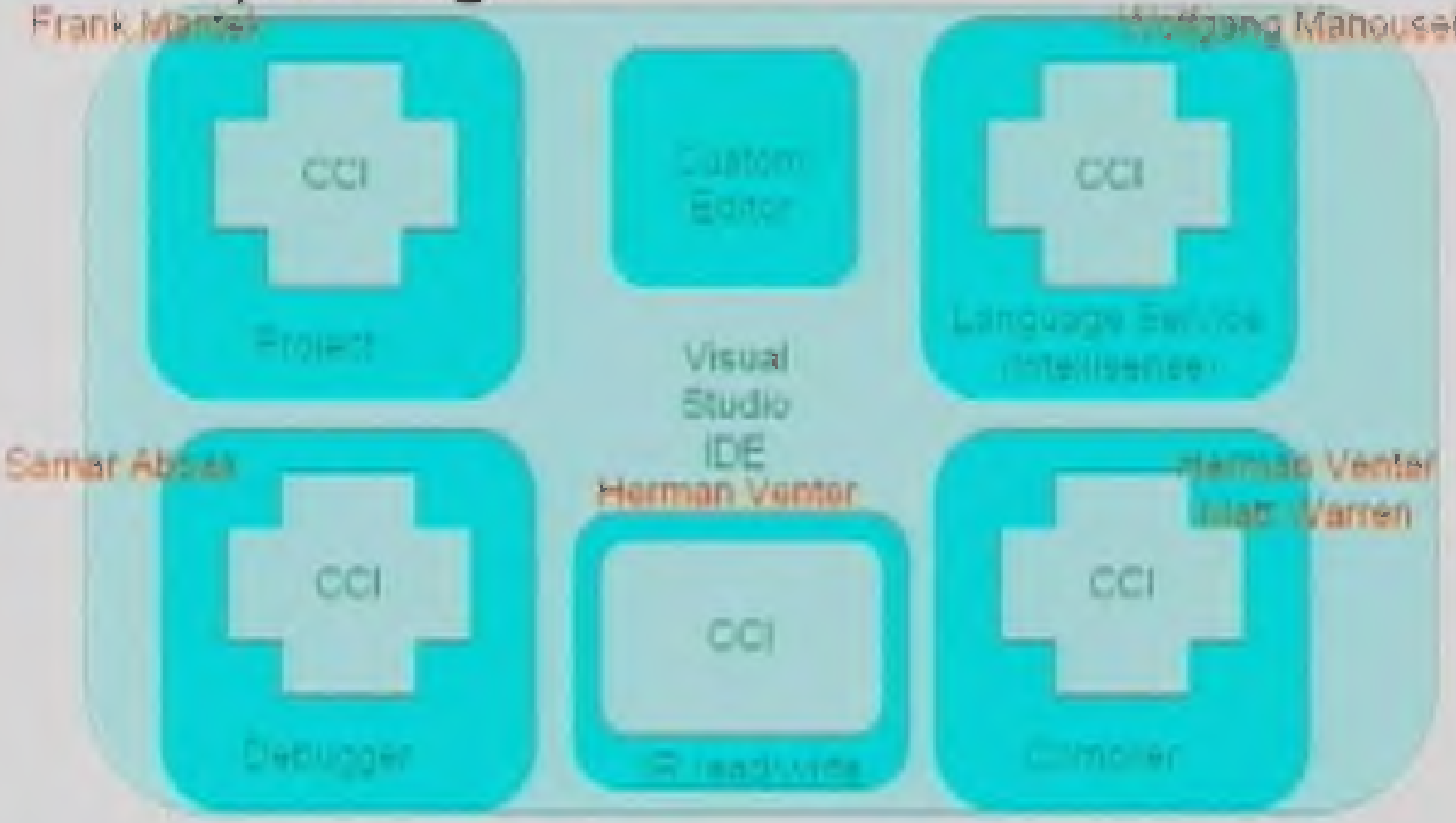
Common Compiler Infrastructure

Not quite a semester project for undergrads

- Better described as a collection of frameworks for writing VS packages



Better described as a collection of frameworks for writing VS packages



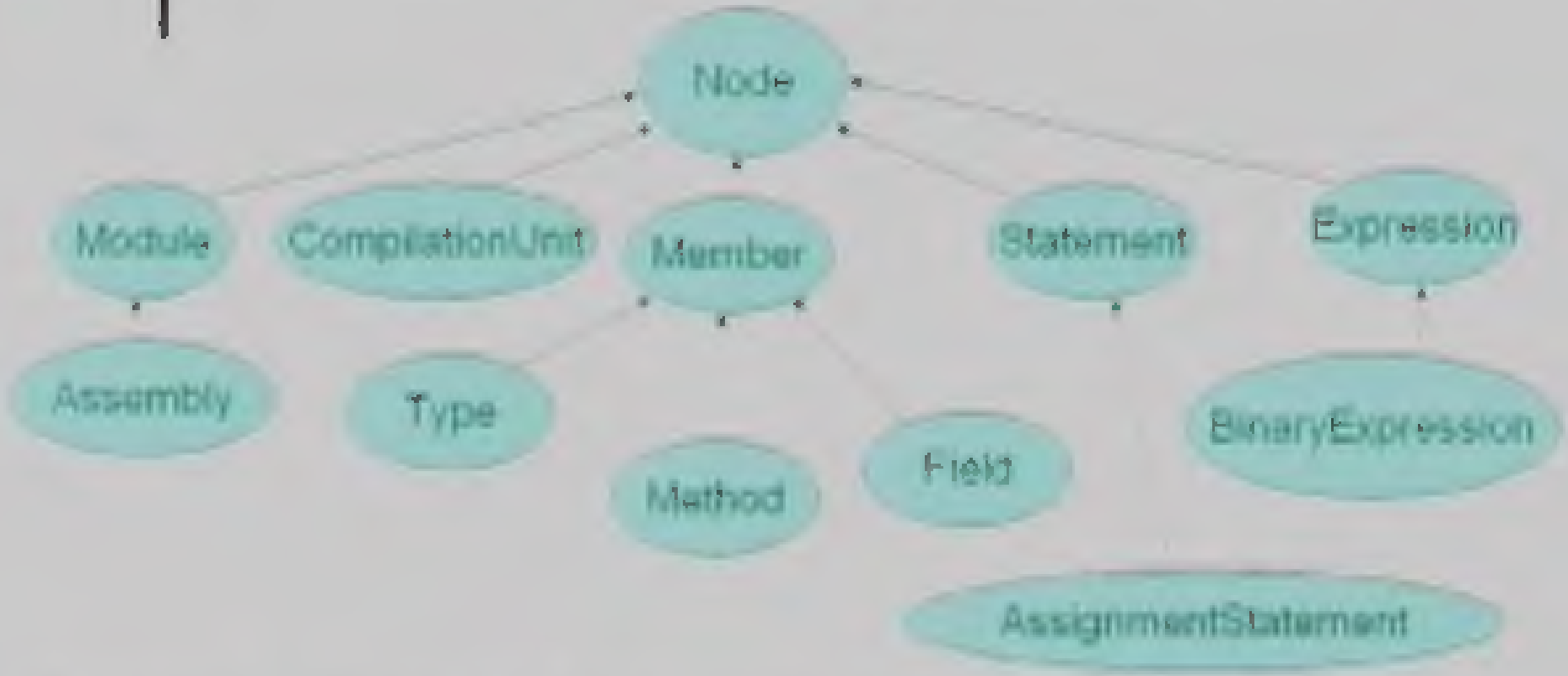
System.Compiler

- *The* class library for writing managed compilers and related tools on the CLI.
- Replaces
  - System.Reflection
  - System.Reflection.Emit
  - System.CodeDom
- Currently packaged as

  System.Compiler.dll

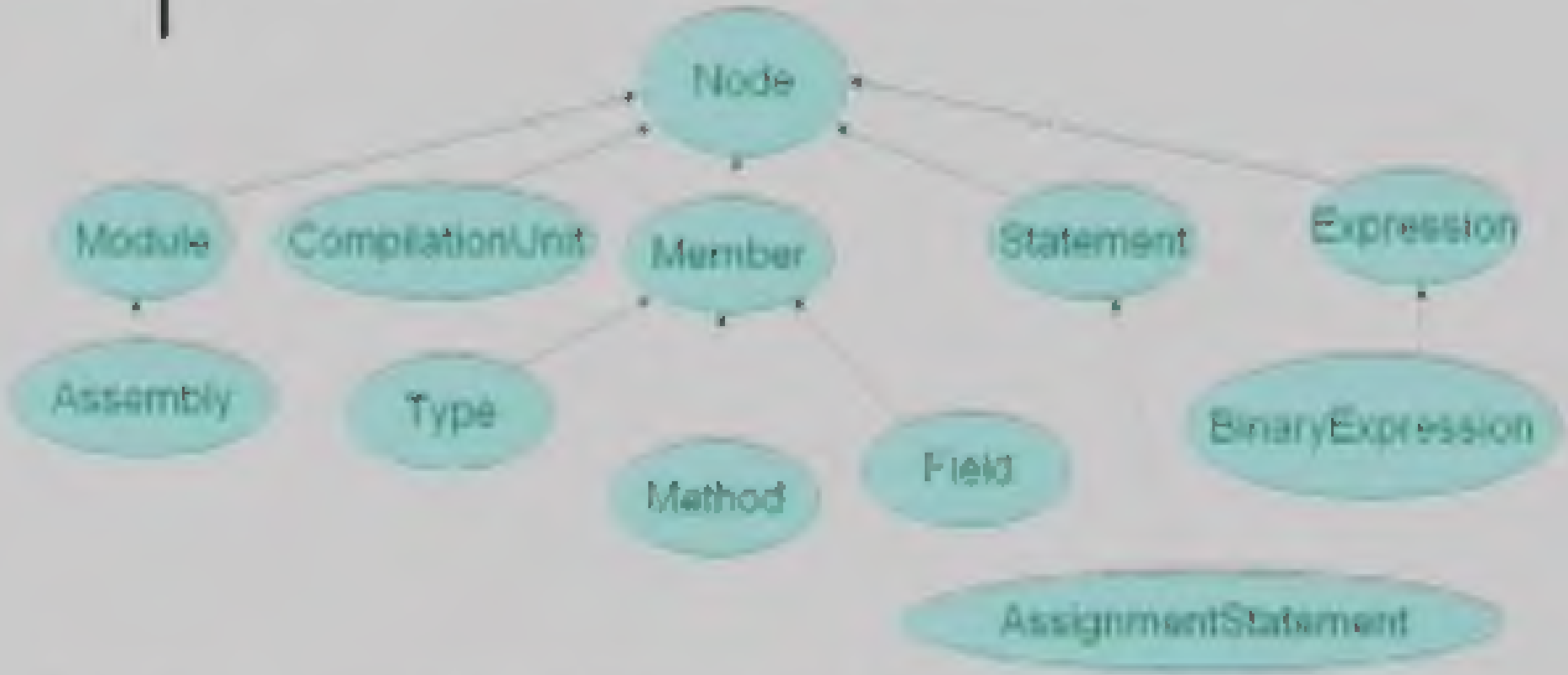
  System.Compiler.Framework.dll

Intermediate Representation



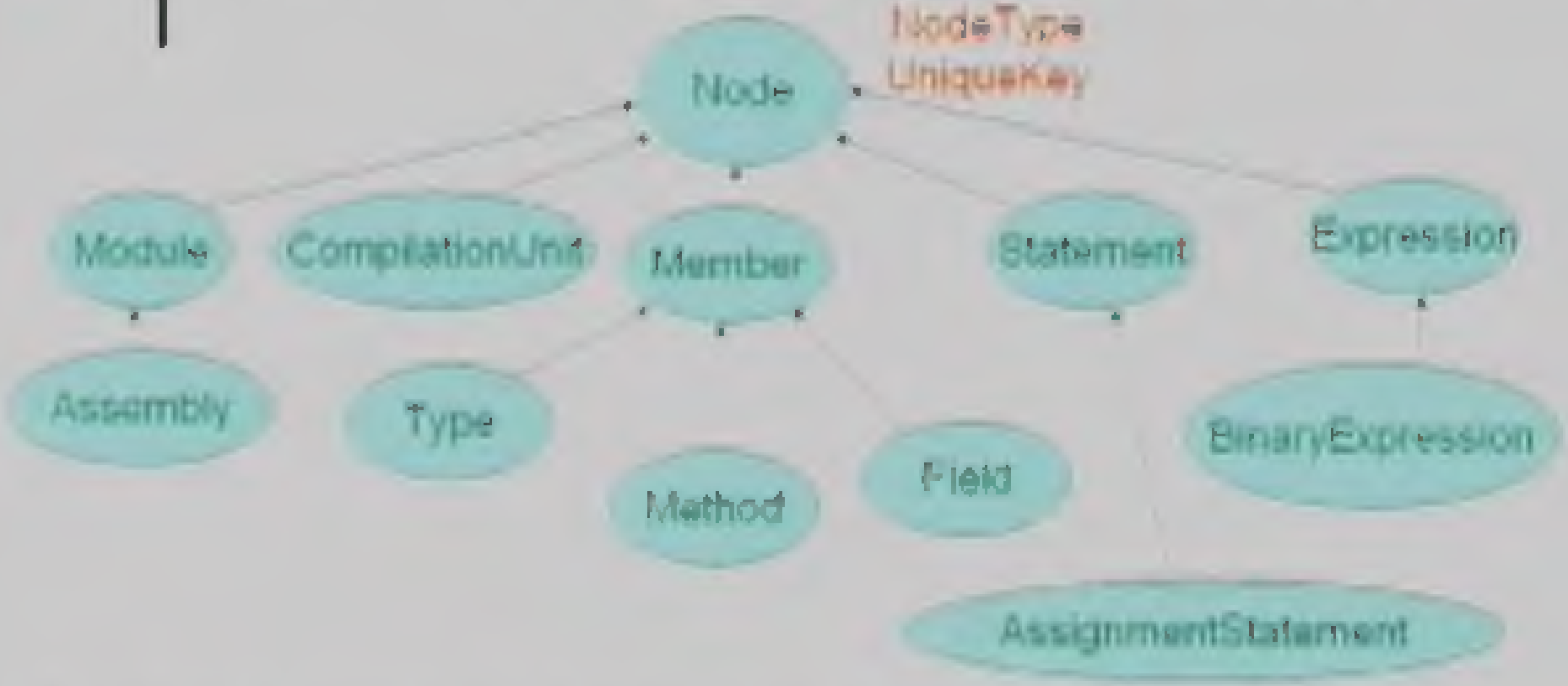
Combines object model for metadata with object model for IL and traditional AST

Intermediate Representation



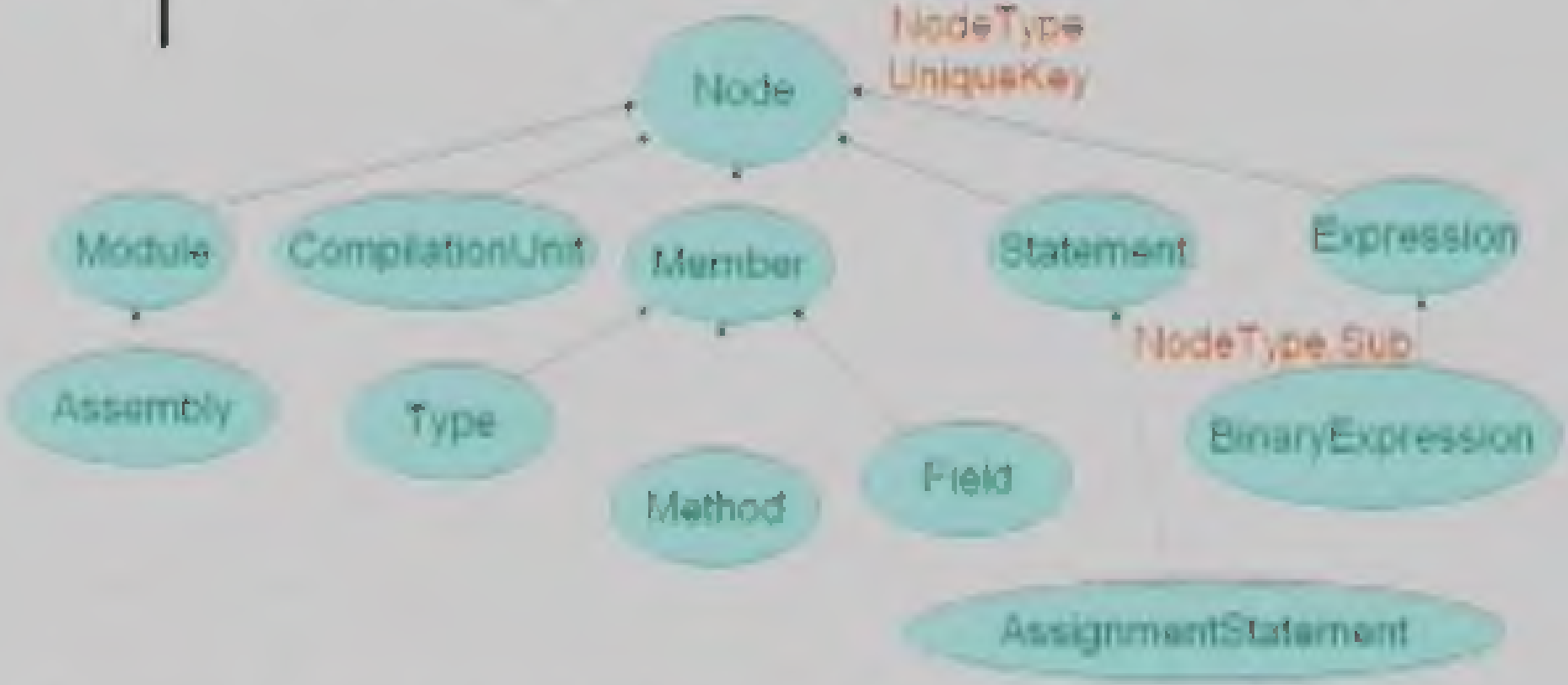
Combines object model for metadata with object model for IL and traditional AST

Intermediate Representation



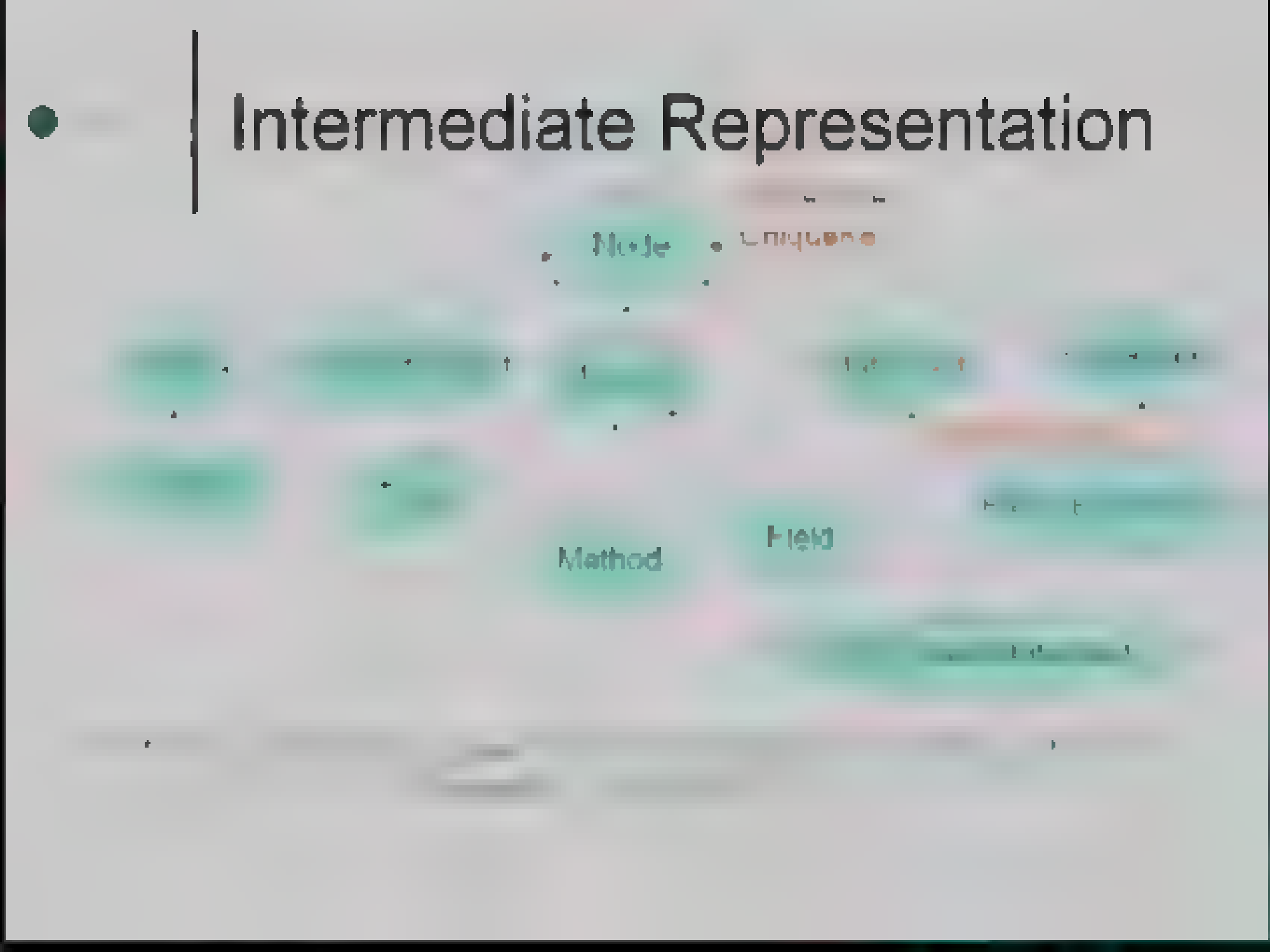
Combines object model for metadata with object model for IL and traditional AST

Intermediate Representation



Combines object model for metadata with object model for IL and traditional AST

Intermediate Representation



Metadata reading and writing

- You can read in an assembly or module by calling Assembly GetAssembly(.) or Module GetModule( )
- You can write out an assembly or module by calling Module WriteModule( )
- You can read or write memory streams not just files
- PDB files also covered see Node SourceContext
- Writer requires IR to be "normalized"

Extending IR into your AST

- Simply extend Node, Statement, Expression, Type, etc. as appropriate.
- Make up your own NodeType enumeration with values that are distinct from System.Compiler.NodeType.
- All fields are public
- All methods are virtual

Visitors

```
case NodeType.AddressDereference
  return this.VisitAddressDereference(
              (AddressDereference)node)

...
public virtual VisitAddressDereference( ){
    ...
    ...
}
```

Visitors

```
case NodeType.AddressDereference
 return this.VisitAddressDereference(
              (AddressDereference)node)

public virtual VisitAddressDereference( ){
}
```

Extending Visitors to cope with your AST nodes

```
default
  return this VisitUnknownNodeType(node)
```

Basic Visitors

- Visitor
- StandardVisitor
- StandardCheckingVisitor
- Duplicator
- Unstacker
- Specializer

Compilation

```
public abstract void Compile(
 CompilationUnit compilationUnit,
 Class globalScope,
 ErrorNodeList errorNodes);
```

Compilation

```
public override void Compile(CompilationUnit compilationUnit,
    Class globalScope, ErrorNodeList errorNodes){
```

Compilation

```
ErrorHandler errorHandler = new ErrorHandler(errorNodes)
                                            , errorHandler );
                                errorHandler ;
                                             ;
                                            , errorHandler ;
                                             ;
```

Compilation

```
looker VisitCompilationUnit(compilationUnit)
```

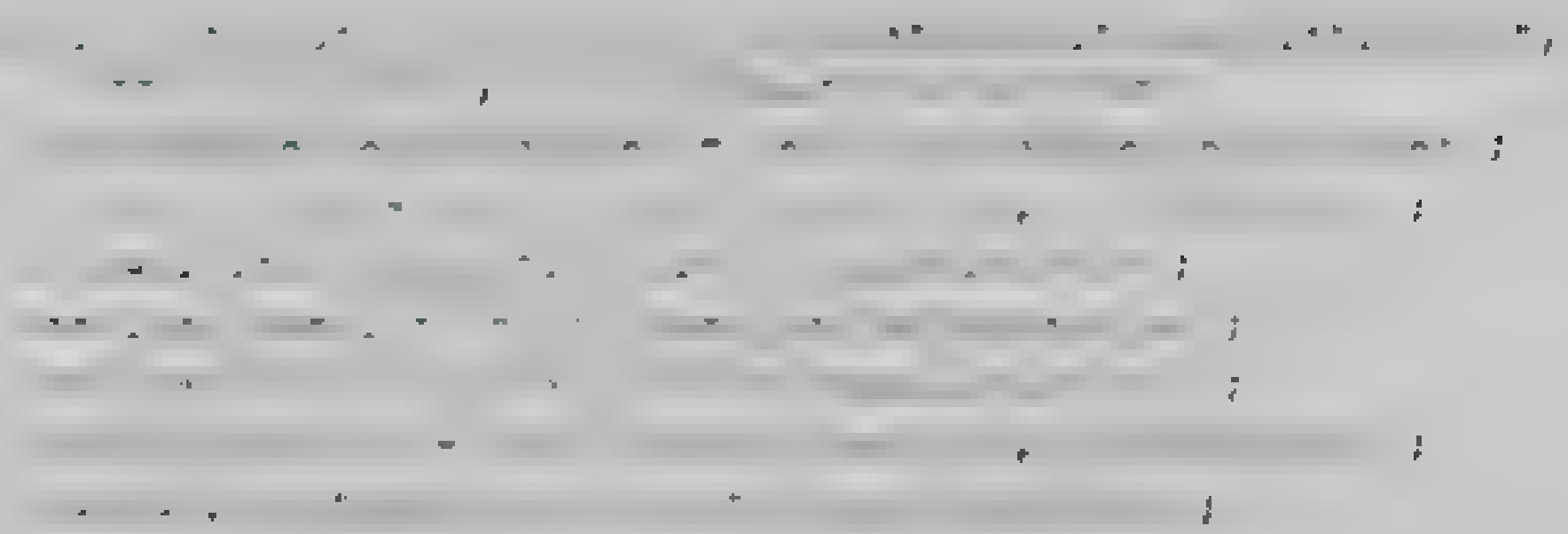
Compilation

```
checker checker = new Checker(resolver  errorHandler)
checker VisitCompilationUnit(compilationUnit)
```

Compilation

```
Normalizer normalizer = new Normalizer()
normalizer VisitCompilationUnit(compilationUnit)
```

Compilation



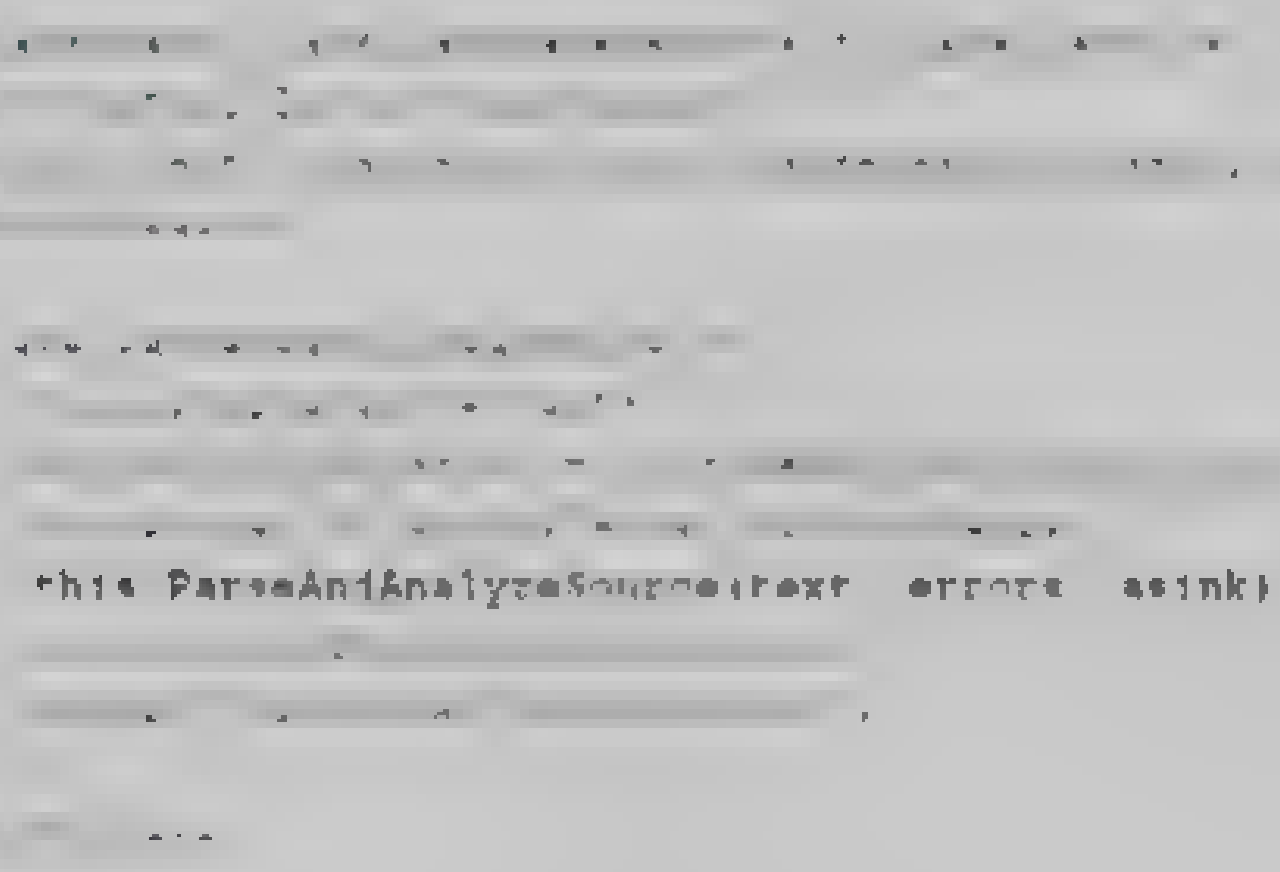
```
Normalizer normalizer = new Normalizer()
normalizer.VisitCompilationUnit(compilationUnit)
```

- Language Service

Language Service

```
public Scanner scanner(
  this.scanner.SetSource(line, 0);
  scanner.ScanTokenAndProvideInfoAboutIt
}
```

Language Service



Language Service

```
public Parser parser;
```

Language Service

```
public Parser parser;
```

Language Service

```
ModuleReferenceList modRefs =
compilationUnit TargetModule ModuleReferences
 for (int i = 0  n = project modules Length  i < n  i++){
   Module m = project modules[i]
   if (m Name == currentFileName) continue
   modRefs Add(new ModuleReference(m Name  m))
 }
```

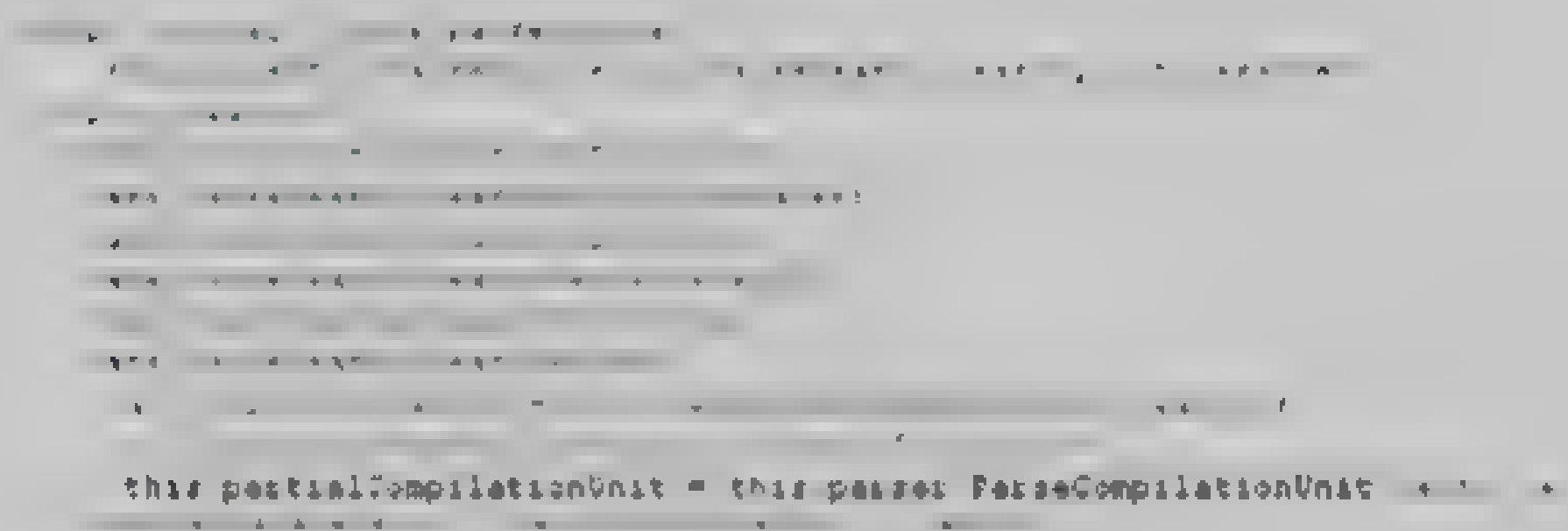
Language Service

```
this AnalyzeParseTree(compilationUnit
this.GetGlobalScope(project)   errors)
compilationUnit TargetModule Name = async FileName
project UpdateWith(compilationUnit TargetModule)
```

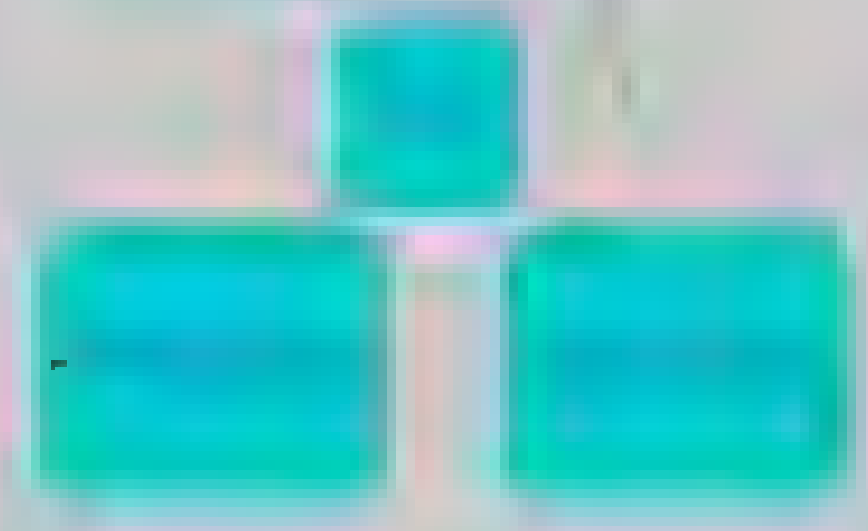
Language Service

```
looker VisitCompilationUnit
resolver VisitCompilationUnit
checker VisitCompilationUnit
```

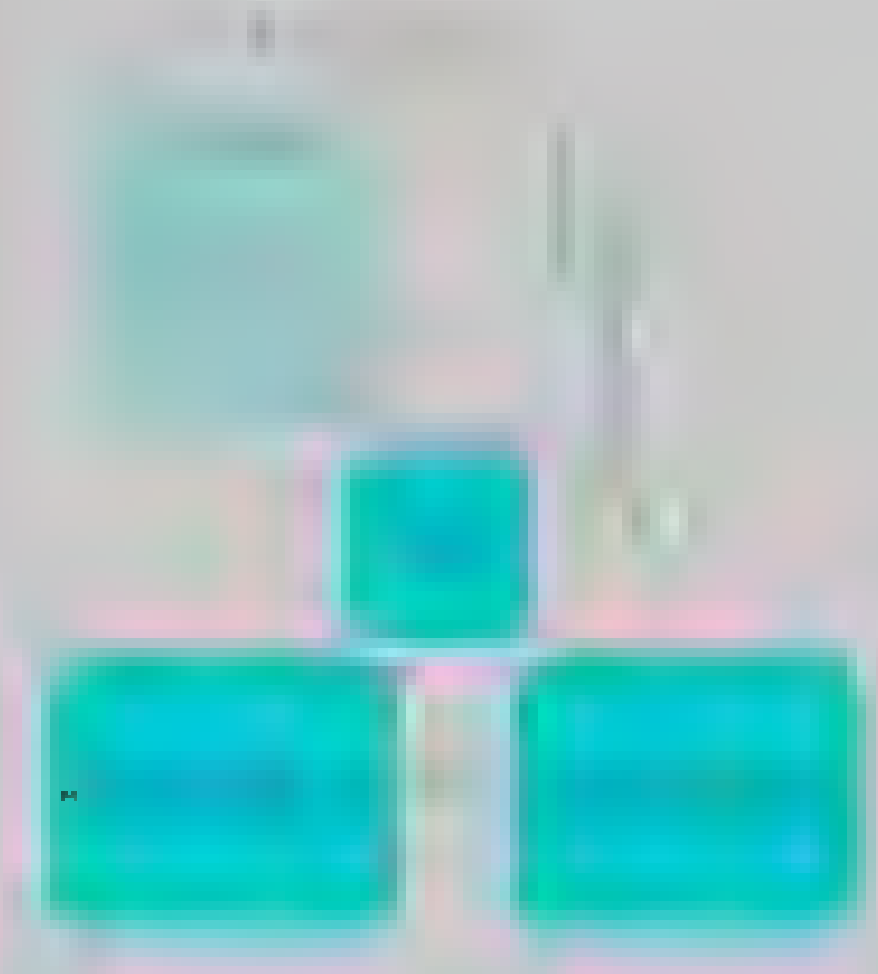
Language Service



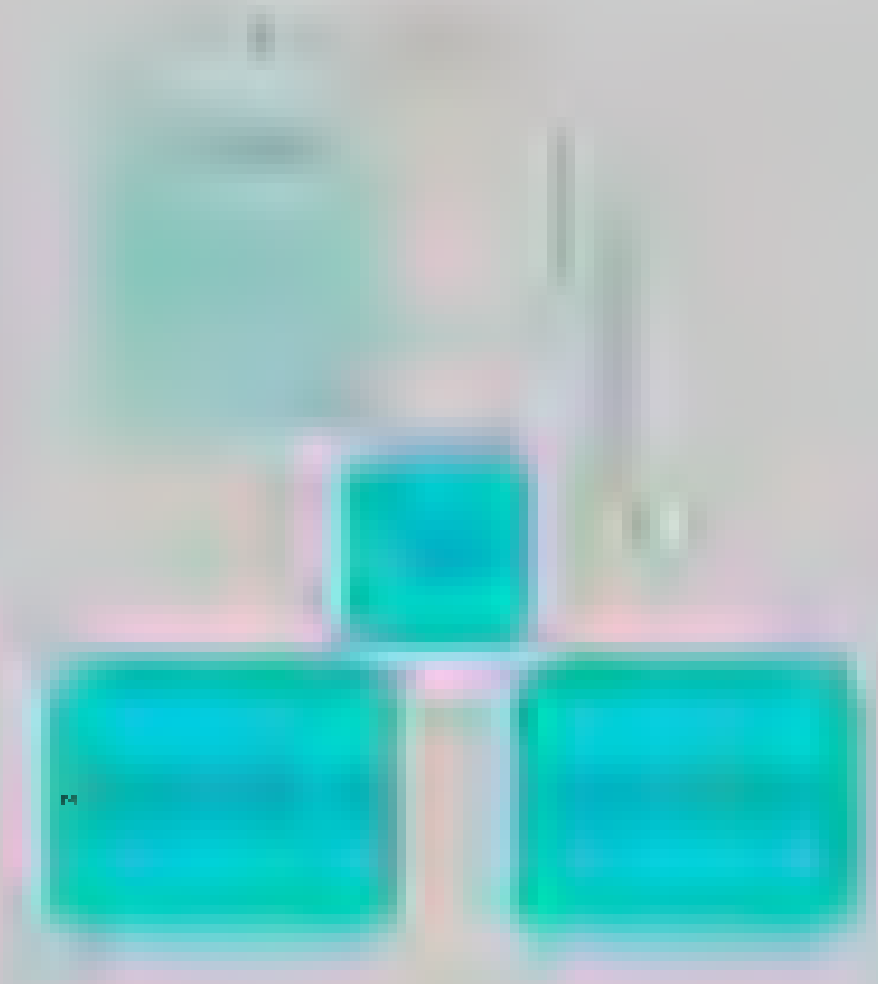
- Language Service



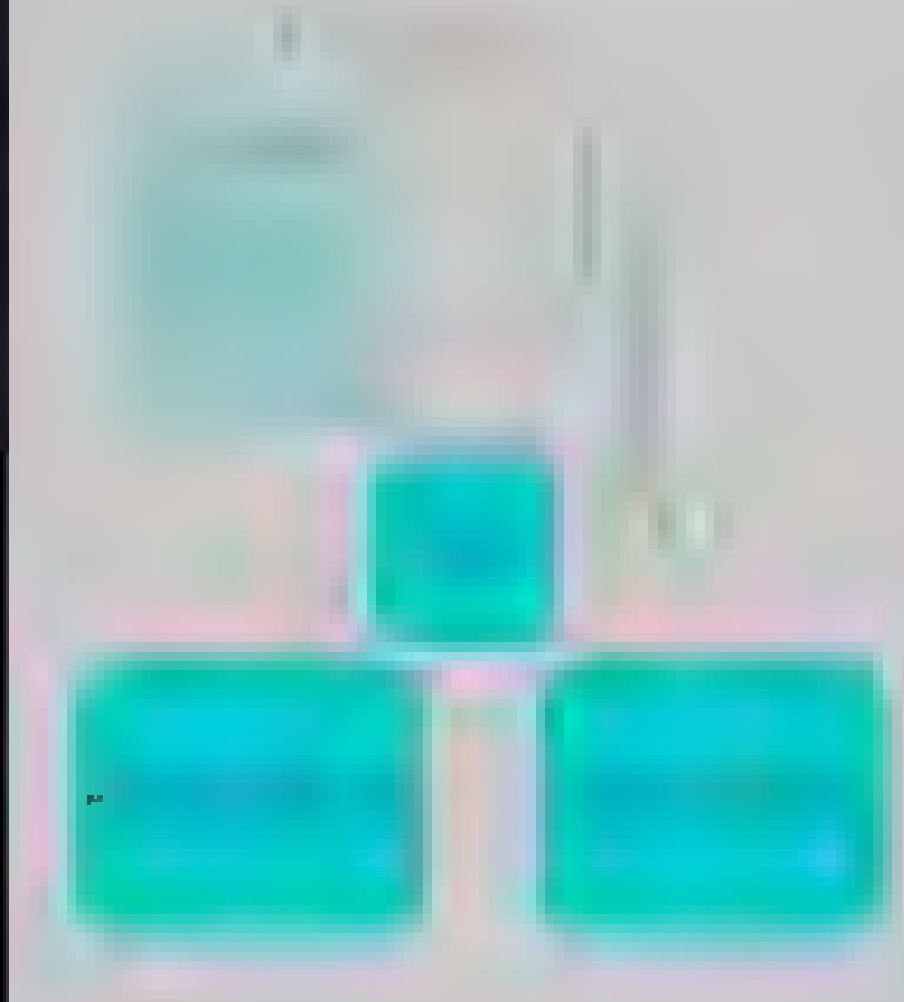
- Language Service



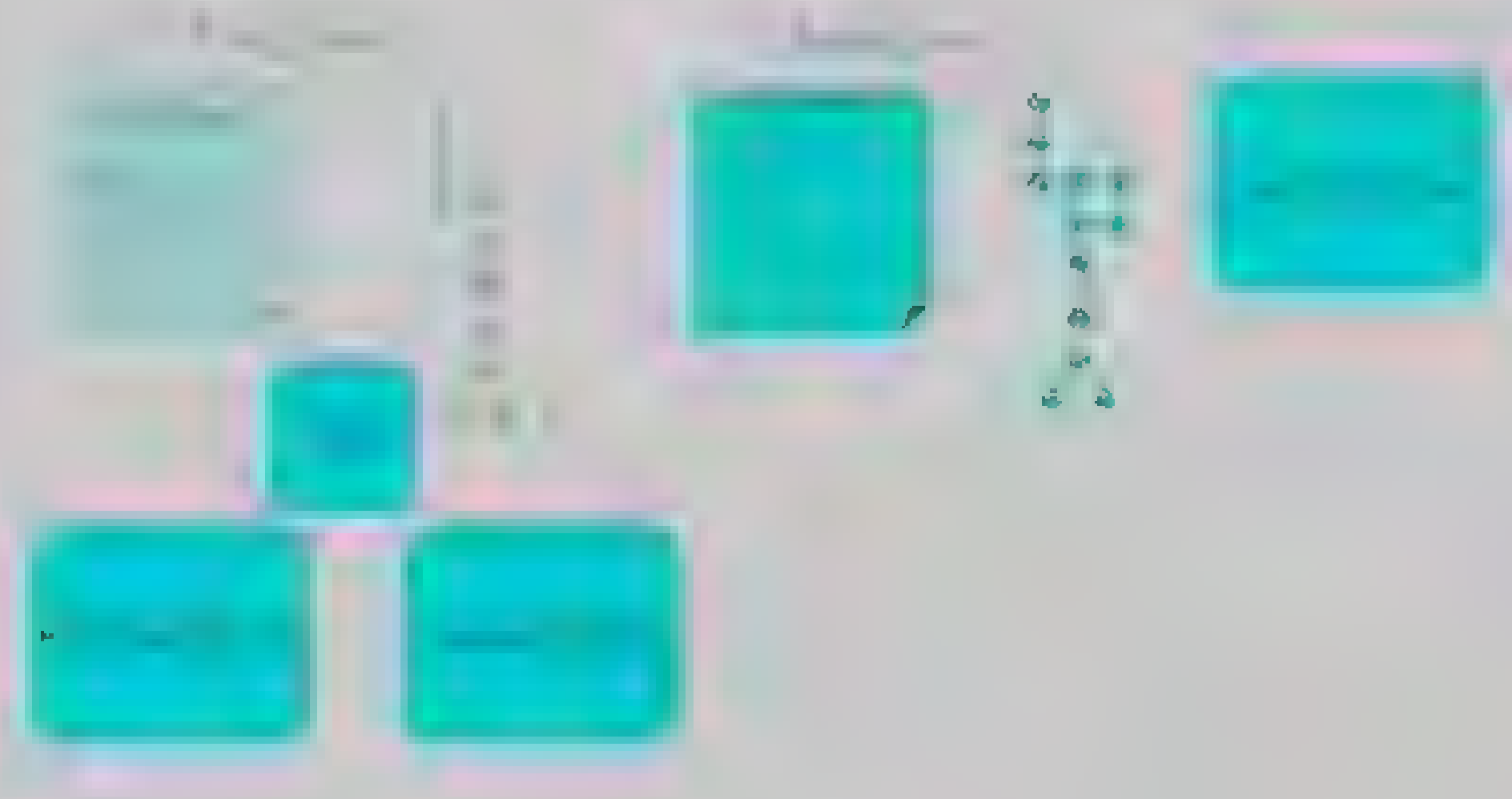
- Language Service



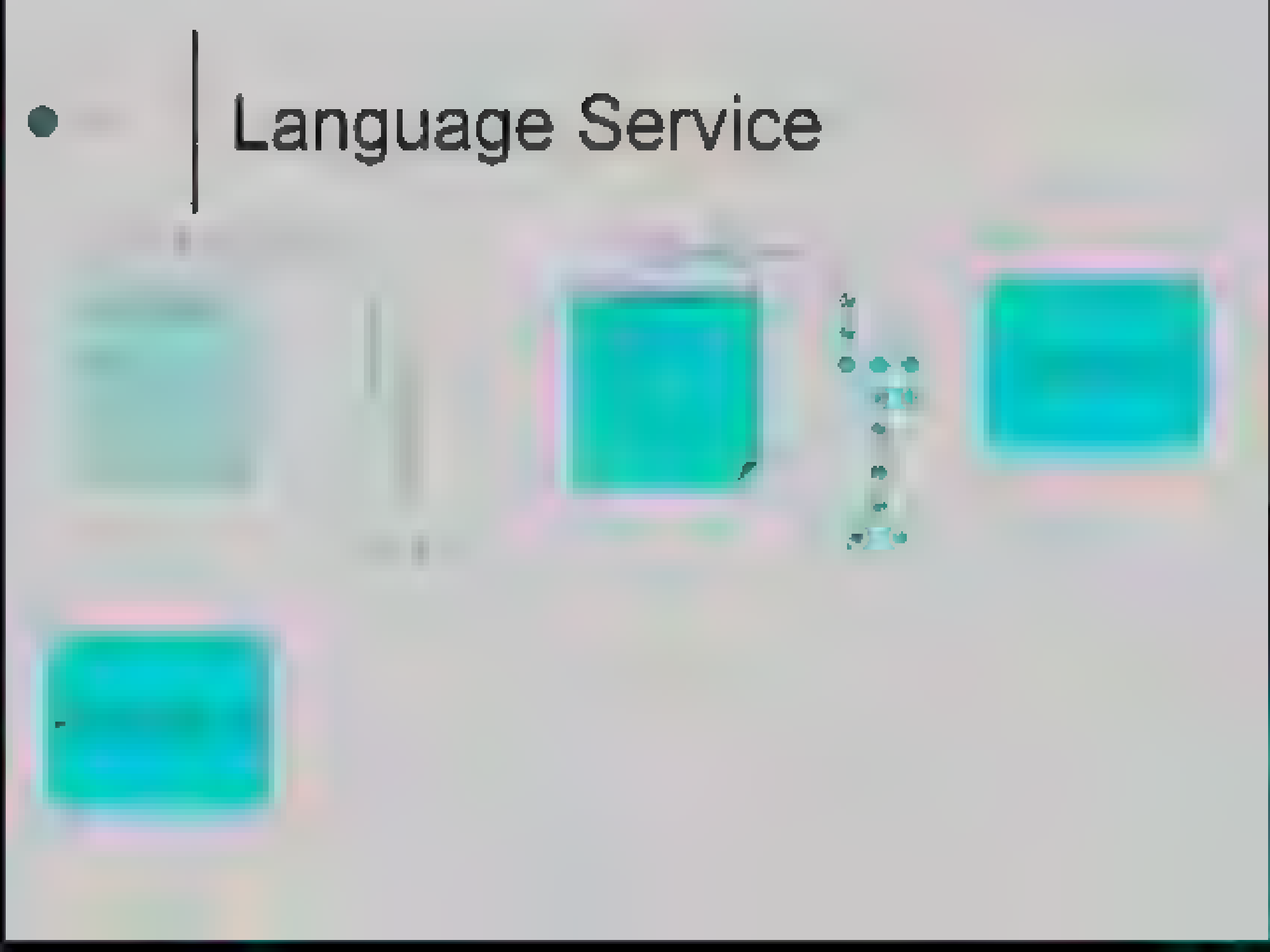
- Language Service



Language Service



- Language Service



Language Service

Language Service

Language Service

- Language Service

Language Service

Project

- Still in early stage of development
- Uses Compiler
- Provides information to Language Service
- Otherwise a huge chunk of code in a world of its own

Debugger

- Uses compiler to compile expressions
- Uses ErrorHandler to format output
- Provides an interpreter for IL, which is used by the compiler to do constant folding.
- This takes care of most language specific issues

Custom Checkers

```
this.RunCustomCheckers(compilationUnit, errorHandler);
```

Custom Checkers

```
this.RunCustomCheckers(compilationUnit, errorHandler);
```

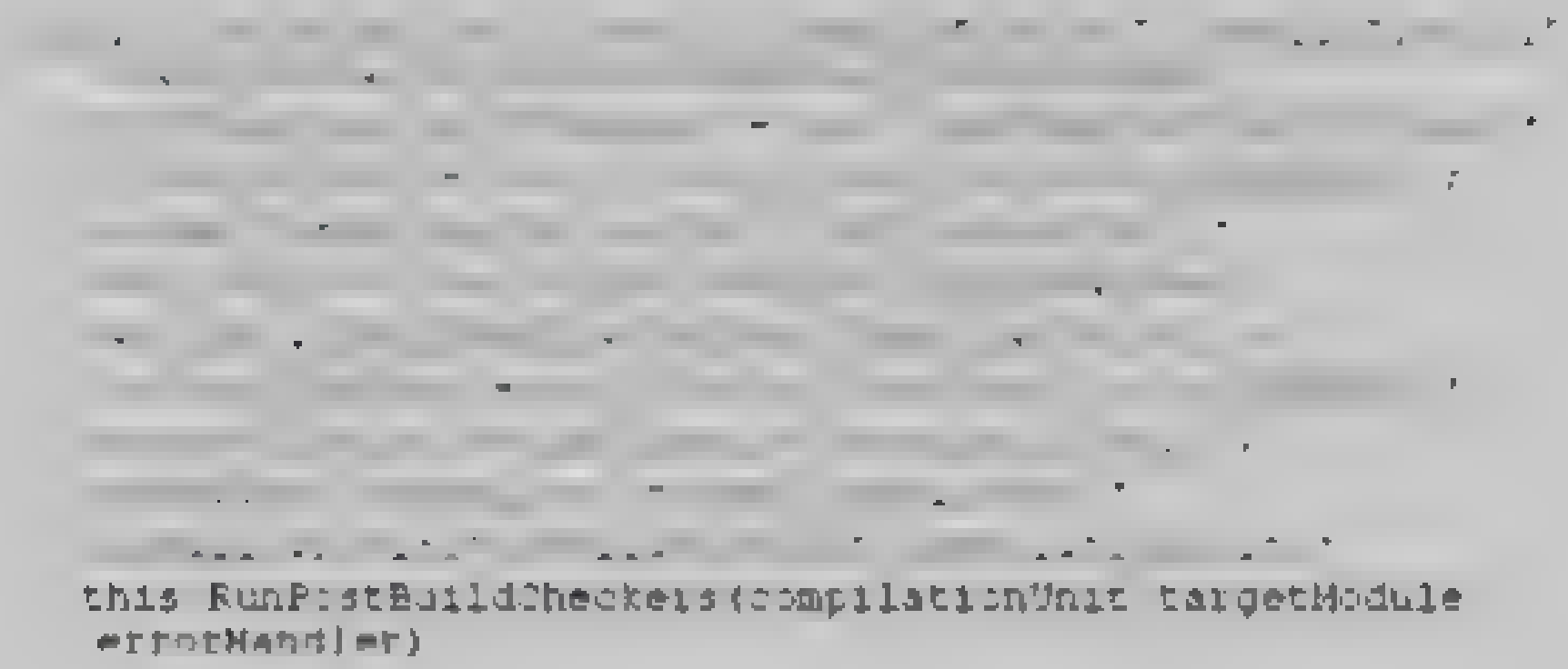
Custom Checkers

```
this.RunCustomCheckers(compilationUnit, errorHandler);
```

Custom Checkers

```
public void RunCustomCheckers(
  CompilationUnit compilationUnit,
  ErrorHandler errorHandler)
{
  foreach (StandardCheckingVisitor cChecker
    in this.GetCustomCheckers(errorHandler))
    cChecker.VisitCompilationUnit(compilationUnit);
}
```

Post Build Checkers



Embedded Foreign Languages

- All CCI languages use the same base types for Type, Expression, Statement, Block, etc.
- This means that IR trees produced by different parsers can be grafted together into a single tree.
- The single tree is compiled co-operatively by separate visitor instances.
- This requires some help from VisitUnknownNodeType.

Embedded foreign languages

```
public virtual Node VisitUnknownNodeType(Node node){
  public virtual Node VisitUnknownNodeType(Node node){
    Visitor visitor = this.GetVisitorFor(node);
    if (visitor == null) return node;
    if (this.callingVisitor != null)
      this.callingVisitor.TransferStateTo(visitor);
    this.TransferStateTo(visitor);
    node = visitor.Visit(node);
    visitor.TransferStateTo(this);
    if (this.callingVisitor != null)
      visitor.TransferStateTo(this.callingVisitor);
    return node;
  }
}
public virtual Visitor GetVisitorFor(Node node){
  return (Visitor)node.GetVisitorFor(this,
   this.GetType().Name);
}
```

Compiler Plugins

```
public override void Compile(CompilationUnit compilationUnit,
    Class globalScope, ErrorNodeList errorNodes){
  ErrorHandler errorHandler = new ErrorHandler(errorNodes);
  Looker looker = new Looker(globalScope, errorHandler);
  looker.VisitCompilationUnit(compilationUnit);
  Resolver resolver = new Resolver(errorHandler);
  resolver.VisitCompilationUnit(compilationUnit);
  Partitioner partitioner = new Partitioner();
  partitioner.VisitCompilationUnit(compilationUnit);
  Checker checker = new Checker(resolver, errorHandler);
  checker.VisitCompilationUnit(compilationUnit);
  Normalizer normalizer = new Normalizer();
  normalizer.VisitCompilationUnit(compilationUnit);
}
```

Links

http://xsharp

mailto:hermanv@microsoft.com

http://pgm/cci

http://research.microsoft.com/programs/europe/events/dotnetcc/Version2/Crash%20Course.ppt

SDPORT=vdinc:9100 root:/xsharp